САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка вставками, выбором, пузырьком

Вариант 9

Выполнила:

Коновалова Кира Романовна

К3139

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024г.

Содержание отчета:

Оглавление

Введение

В данной лабораторной работе я реализовала и протестировала несколько алгоритмов сортировки на языке Python.

Задачи по варианту

Задача 1. Сортировка вставками

Формулировка задачи:

Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива A = {31, 41, 59, 26, 41, 58}.

Код программы:

```
'''Задание 1. Сортировка вставкой'''
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, massive = f.readlines()
```

```
    array = insertion_sort(list(map(int, massive.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))), file=f)
```

Описание работы алгоритма:

Алгоритм последовательно перебирает элементы массива начиная со второго элемента, так как первый уже отсортирован. В цикле, если j не отрицателен и до тех пор, пока текущий элемент меньше предыдущего, мы сдвигаем предыдущий элемент вправо и затем вставляем текущий элемент. Таким образом массив упорядочивается по возрастанию.

Тестирование:

```
import unittest

from lab1.task1.src.isertion_sort import insertion_sort

class TestInsertionSort(unittest.TestCase):
    def test_insertion_sort(self):
        self.assertEqual(insertion_sort([31, 41, 59, 26, 41, 58]), [26,
31, 41, 41, 58, 59])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

Алгоритм сортировки вставками успешно мною реализован и протестирован. Сложность алгоритма составляет O(n**2). Из этого мы можем сделать вывод, что он не очень эффективен для больших массивов. Но для небольших массивов он работает довольно эффективно.

Задача 2. Сортировка вставками с отслеживанием индексов

Формулировка задачи:

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

Код программы:

```
"""Задание 2. Сортировка вставкой +"""
def insertion_sort(list_arr):
    index_result = [1]
    for i in range(1, len(list_arr)):
        for j in range(i - 1, -1, -1):
            if list_arr[i] < list_arr[j]:
```

```
                list_arr[i], list_arr[j] = list_arr[j], list_arr[i]
                i, j = j, i
        index_result.append(i + 1)
    return index_result, list_arr
if __name__ == '__main__':
    with open('input.txt') as f:
        n, mas = f.readlines()
    ind, array = insertion_sort(list(map(int, mas.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, ind))), file=f)
        print(' '.join(list(map(str, array))), file=f)
```

Описание работы алгоритма:

Алгоритм работает аналогично алгоритму сортировки вставками из первой задачи, но одновременно с этим он ведет массив индексов, который отслеживает исходные позиции элементов.

В результате, после работы алгоритма мы получаем отсортированный массив и список индексов, исходных позиций элементов.

Тестирование:

```
import unittest
from lab1.task2.src.insertion_sort_index import insertion_sort
class TestInsertionSort(unittest.TestCase):
    def test_insertion_sort_index(self):
        ind, arr = insertion_sort([1, 8, 4, 2, 3, 7, 5, 6, 9, 0])
        self.assertEqual(arr, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        self.assertEqual(ind, [1, 2, 2, 2, 3, 5, 5, 6, 9, 1])
if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

Данный алгоритм успешно сортирует массив и возвращает список исходных индексов элементов.

Задача 3. Сортировка вставками по убыванию

Формулировка задачи:

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swap.

Код программы:

```
def insertion_descending(arr):
    for i in range(1, len(arr)):
        key = arr[i]
```

```
        j = i - 1
        while j >= 0 and key > arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

if __name__ == '__main__':
    with open('input.txt') as f:
        n, mas = f.readlines()
    array = insertion_descending(list(map(int, mas.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))), file=f)
```

Описание работы алгоритма:

Данный алгоритм работает аналогично обычному алгоритму сортировки по возрастанию, за исключением смены знака больше (>) на меньше (<), чтобы осуществить сортировку по убыванию.

Тестирование:

```
import unittest
from lab1.task3.src.insertion_descending import insertion_descending
class TestInsertionDescendingSort(unittest.TestCase):
    def test_insertion_descending(self):
        self.assertEqual(insertion_descending([31, 41, 59, 26, 41, 58]),
[59, 58, 41, 41, 31, 26])
if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

Алгоритм сортировки вставками по убыванию успешно работает

Задача 5. Сортировка выбором

Формулировка задачи:

Написать алгоритм сортировки выбором

Код программы:

```
'''Задача 5. Сортировка выбором.'''
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

if __name__ == '__main__':
```

```
    with open('input.txt') as f:
        n, mas = f.readlines()
    array = selection_sort(list(map(int, mas.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))), file=f)
```

Описание работы алгоритма:

Данный алгоритм делит массив на две части – отсортированную и неотсортированную. В цикле находиться минимальный элемент из неотсортированной части и ставиться в самое начало.

Тестирование:

```
import unittest

from lab1.task5.src.selection_sort import selection_sort

class TestSelectionSort(unittest.TestCase):
    def test_selection_sort(self):
        self.assertEqual(selection_sort([31, 41, 59, 26, 41, 58]), [26,
31, 41, 41, 58, 59])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

Алгоритм сортировки выбором успешно мною реализован и протестирован. Сложность алгоритма составляет O(n**2)

Задача 6. Пузырьковая сортировка

Формулировка задачи:

Напишите код на Python и докажите корректность пузырьковой сортировки.

Код программы:

```
'''Задание 6. Пузырьковая сортировка.'''
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

```
if __name__ == '__main__':
    with open('input.txt') as f:
        n, mas = f.readlines()
    array = bubble_sort(list(map(int, mas.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))), file=f)
```

Описание работы алгоритма:

Данный алгоритм последовательно сравнивает значения соседних элементов и меняет числа местами, если предыдущее число оказывается больше последующего. И так до полной сортировки массива.

Тестирование:

```
import unittest
from lab1.task6.src.bubble_sort import bubble_sort

class TestBubbleSort(unittest.TestCase):
    def test_bubble_sort(self):
        self.assertEqual(bubble_sort([31, 41, 59, 26, 41, 58]), [26, 31,
41, 41, 58, 59])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

Алгоритм пузырьковой сортировки успешно мною реализован и протестирован.
Сложность данного алгоритма составляет O(n**2), поэтому он будет неэффективен на больших массивах.

Заключение

В ходе выполнения данной лабораторной работой мною были реализованы следующие алгоритмы: сортировка вставками (по возрастанию, по убыванию и с отслеживанием индексов), сортировка выбором и пузырьковая сортировка

Все эти алгоритмы имеют сложность O(n2), что делает их простыми в реализации, но не очень эффективными для больших массивов.**